基于分块存储格式的稀疏线性系统求解优化 *

程　凯，田　瑾，吴　飞，汪　茹，李洪芹

(上海工程技术大学 电子电气工程学院，上海 201620)

摘　要：针对基于 GPU 求解大规模稀疏线性方程组进行了研究，提出一种稀疏矩阵的分块存储格式 HMEC (hybrid multiple ELL and CSR)。通过重排序优化系数矩阵的存储结构，将系数矩阵以一定的比例分块存储，采用 ELL 与 CSR 存储格式相结合的方式以适应不同的分块特征，分别使用适用于不对称矩阵的不完全 LU 分解预处理 BICGStab 法和对称正定矩阵的不完全 Cholesky 分解预处理共轭梯度法求解大规模稀疏线性系统。实验表明，应用 HMEC 格式存储稀疏矩阵并以调用 GPU kernel 的方式实现前述两种方法，与其他存储格式的实现方式作比较，最优可分别获得 31.89% 和 17.50%的加速效果。

关键词：GPU 加速；共轭梯度；稳定双共轭梯度；重排序；HMEC 存储格式；稀疏矩阵与向量乘

中图分类号：TP391　　doi: 10.3969/j.issn.1001-3695.2018.04.0284

Optimization of solving sparse linear system based on blocked storage format

Cheng Kai, Tian Jin, Wu Fei, Wang Ru, Li Hongqin

(*College of Electronic & Electrical Engineering*, *Shanghai University of Engineering Science*, *Shanghai* 201620, *China*)

Abstract: This paper proposed a storage format HMEC (Hybrid Multiple ELL and CSR) of sparse matrix to solve large sparse linear equations on GPU. Firstly, we optimized the storage structure of the coefficient matrix by reordering. Secondly, we stored the coefficient matrix in a certain scale block. Then we adopt an approach by combining ELL and CSR storage format to adapt to different characteristics of blocks. At last, we took Bi-Conjugate Gradient Stabilized (BICGStab) and Conjugate Gradient (CG) iterative methods to solve large sparse linear systems, they are respectively preconditioned by incomplete-LU and incomplete-Cholesky factorization for asymmetric and symmetric positive definite linear matrices. Experiments show that comparing the way by storing sparse matrices in HMEC format with other ways by storing in the common storage format, the acceleration of the best available we can get are 31.89% and 17.50%.

Key words: GPU acceleration; conjugate gradient; bi-conjugate gradient stabilized; reorder; HMEC (hybrid multiple ELL and CSR) storage format; sparse matrix-vector multiplication

0　引言

求解大规模稀疏线性系统广泛应用于计算科学与工程的各个领域[1,2]。目前，常用的求解稀疏线性系统的方法有直接法与迭代法[3,4]，其中，迭代法尤为适合大规模稀疏线性系统的求解。常用的迭代法包括固定的迭代法，如基于分割的 Jacobi 和 Gauss-Seidel (GS)法;不固定的迭代法，如 Krylov 子空间方法，包括可适用于不对称系统的稳定双共轭梯度(bi-conjugate gradient stabilized，BICGSTAB) [5][6][7]法和对称正定线性系统的共轭梯度(conjugate gradient)法[8]。在实践中，当稀疏矩阵为病态矩阵时，需用预处理技术加以改进以达到良好的计算效果。

Mayer [9]研究了基于多级不完全 LU 分解预处理法求解线性三角系统；Naumov[10]在 Mayer 的基础上研究运用 Cubals[11]和 Cusparse[12]库的迭代法求解大型方程组；殷建[13]提出了一种基于 HYB[14](Hybrid)格式的混合存储格式，通过持续划分 ELL 以提升性能；阳王东等人针对 SpMV 的优化提出了一种重排序的分块存储格式 BCE[15]，良好地适应了有不规则非零元结构的矩阵。本文在阳王东的研究基础上将系数矩阵 *A* 重新排序，在殷建的研究上采用多次划分 ELL 格式以减弱并行计算时的负载不平衡，并采用 CSR 格式存储剩余矩阵以提升存储效率。最终实验表明，采用本文所提出的分块存储格式对于大规模稀疏线性系统的求解具有良好的加速效果。

1　稀疏矩阵的存储格式

因稀疏矩阵中非零元素分布的不规则性，导致采用单一的存储格式进行运算时，取得的效果并不理想，鉴于此，多种优

收稿日期：2018-04-19；**修回日期：**2018-05-27　**基金项目：**国家自然科学基金资助项目（61272097）；上海市自然科学基金资助项目（15ZR1418900）

作者简介：程凯（1993-），男，硕士研究生，主要研究方向为面向电磁问题的大规模并行计算（ck1625026036@163.com）；吴飞，教授，博士，主要研究方向为计算机信息处理与节能控制；汪茹，硕士研究生，主要研究方向为大规模并行计算；李洪芹，副教授，博士，主要研究方向为数字信息处理与控制.

化矩阵存储格式的方法相继提出。一种是对行合并分块以缓解各行间非零元数差异过大，如分块的 CSR 方法(BCSR) [16][17]、(BELLPACK) [18] 格式等，分片的 ELLPACK 格式(SELL-PACK)[19]等；另一种是采取混合格式进行存储，以适应不规则的稀疏特征，如混合 ELL(ELLPACK)和 COO(Coordinate)格式的 HYB[错误!未定义书签。]存储格式等。

本文所提出的稀疏矩阵分块混合存储格式 HMEC，由多个 ELL 块和一个 CSR 块混合而成，将一个大规模稀疏矩阵分解成多个子矩阵分别计算，并采用重新排序的方法改善稀疏矩阵的非零元结构。其中，CSR 块存储分割出的离散点，良好地适应了系数矩阵的不规则性，极大的提升了稀疏矩阵的存储效率。

2 ELL 存储格式

如图 1 所示，ELL 格式用两个数组 values 和 col 来存储一个 4×2 的矩阵。其中，数组 values 存储原稀疏矩阵中每行非零元的值；数组 col 存储相应非零元素在原系数矩阵中的列索引。

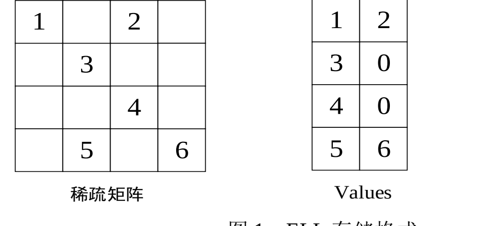


图 1 ELL 存储格式

2.1 CSR 存储格式

如图 2 所示，CSR 格式采用三个数组 value、rowptr、colind 来存储一个稀疏矩阵，数组 values 保存原稀疏矩阵中的所有非零元的值，数组 colind 存储了各非零元在原稀疏矩阵中的列索引，数组 rowptr 存储了每一行元素在压缩存储格式中的起始位置。

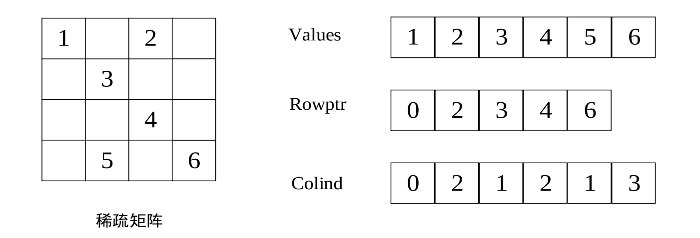


图 2 CSR 存储格式

2.2 HYB 存储格式

如图 3 所示，为便于简述，以一个 6 阶方阵 *A* 为例，HYB 存储格式依据阈值 *K* 将矩阵划分为 ELL 格式与 COO 格式两部分存储，每个格子代表矩阵的一个元素，空格代表 0。ELL 格式存储矩阵 *A* 中非零元的值和它所在的列，COO 格式存储划分后剩余矩阵的值、行和列，并分别用三个数组 values、row 和 colind 表示。

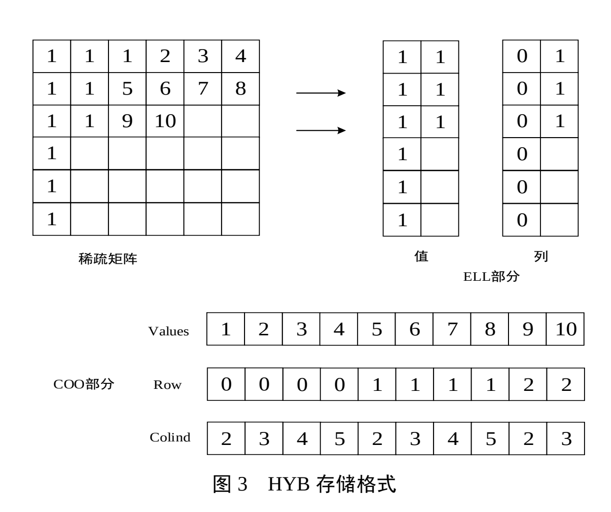


图 3 HYB 存储格式

2.3 HMEC 存储格式

本文提出的 HMEC 存储格式由多个 ELL 块和一个 CSR 块混合而成,即将经过重排序的矩阵 *A* 在未达到停止划分的条件前，根据阈值 *K* 持续划分为 ELL 阵和剩余矩阵，最终将矩阵 *A* 保存为多个以 ELL 格式存储的矩阵块和一个以 CSR 块格式存储的矩阵块，每个格子代表矩阵的一个元素，空格代表 0。为便于简述，以划分 2 次 ELL 为例，如图 4 所示。

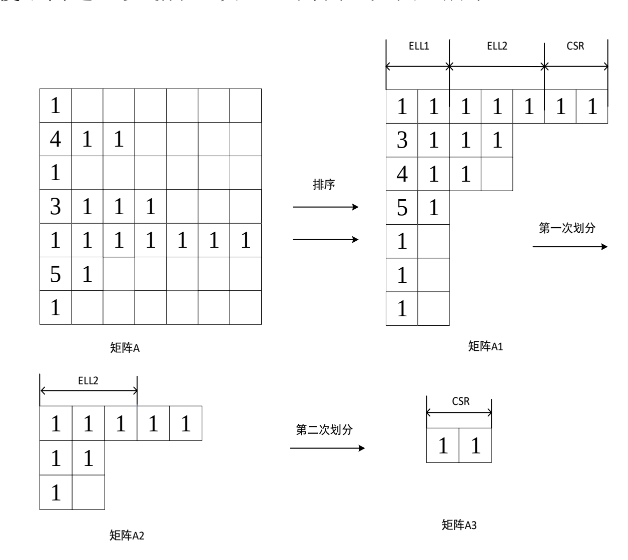


图 4 HMEC 存储格式

对于系数矩阵 *A*，为优化矩阵 *A* 内的非零元分布，采用选择排序法按每行非零元的个数由高至低进行降序排序，并额外用一个矩阵 recoder 记录下排序时的行索引，可得矩阵 *A*1。将矩阵 *A*1 依据阈值 *K* 划分为一个 ELL 格式存储的矩阵 ELL1 和剩余矩阵 *A*2，将矩阵 *A*2 依据阈值 *K* 划分为一个 ELL 格式存储的矩阵 ELL2 和剩余矩阵 *A*3，划分完毕后，矩阵 *A*3 以 CSR 格式存储，最终划分为 ELL1、ELL2 和 CSR 三部分并分别以相应的格式存储，阈值 *K* 的划分规则如下：

求出矩阵 S 中第一列的非零元数 x（第一次划分 S 为 *A*1，第二次划分 S 为 *A*2. . . ）

```
if(x > 512){
for j=2: 1: sf
% j 为矩阵 S 的列，sf 为矩阵 S 的总列数
  求出前 j 列的总非零元数 s ；
```

求出前 j 列的元素的总数量 c ；

若 s > (2c/3), 则 K=j; 否则 break, 跳出循环 ；

求出阈值 K (矩阵为 *A*1 时，*K* 取 *K*1，矩阵为 *A*2 时，*K* 取 *K*2…)

}

其中，第一列的非零元数 *x*>512 为可以持续划分的条件，即剩余矩阵中非零元素的行数少于 512 时会停止划分，因为此时若继续划分已经不能够再获得性能的提升。

3 方程组求解方法

3.1 共轭梯度法

共轭梯度法的研究基础为构造搜索方向和步长因子从而找到合适的下降方向，再进行迭代计算，最终找到最优解并给出收敛性。

具体算法可描述为：对于无约束优化问题，给出一个初始值 x_0，算法迭代产生 $x_1, x_2,…, x_k$，若某一 x_k 是优化问题的解，在第 *k* 次迭代时，当前迭代点为 x_k，则线搜索型的方法产生一个搜索方向 p_k 后，下一个迭代点 x_{k+1} 的计算公式为：$x_{k+1}=x_k+\alpha_k p_k$，其中 α_k 是步长因子，它满足某些线搜索条件。

3.2 不完全 Cholesky 分解

对于方程组

$$Ax=b \tag{1}$$

其中：*A* 为 *n* 阶对称正定矩阵，*b* 为列向量，*x* 为待求解。若对矩阵 *A* 进行不完全 Cholesky 分解，可得式(2)，其中 *L* 为下三角矩阵，*D* 为对角矩阵，误差矩阵 *E* 为矩阵 *A* 和 LDL^T 之差。若矩阵 *A* 中零元素 $a_{ij}=0$，可人为地令矩阵 *L* 中 $l_{ij}=0$，以使矩阵 *L* 的稀疏性与矩阵 *A* 一致。

$$A=LDL^T+E \tag{2}$$

若用矩阵 $(LD^{1/2})^{-1}$ 左乘公式(1)，可得式(3)。

$$(LD^{1/2})^{-1}A\left((LD^{1/2})^T\right)^{-1}(LD^{1/2})^T x=(LD^{1/2})^{-1}b \tag{3}$$

令式(3)中 $(LD^{1/2})^T=\tilde{R}$，其中，$\tilde{R}$ 为上三角矩阵，可得预处理矩阵 *M* 如式(4)所示。

$$A\approx M=\tilde{R}^T\tilde{R} \tag{4}$$

式(3)可简化为式(5)。

$$(\tilde{R}^{-T}A\tilde{R}^{-1})(\tilde{R}x)=(\tilde{R}^{-T}b) \tag{5}$$

令式(5)中 $(\tilde{R}^{-T}A\tilde{R}^{-1})=C$，$\tilde{R}x=y$，$(\tilde{R}^{-T}b)=p$，可得式(6)。

$$Cy=p \tag{6}$$

式(6)中 *C* 即为经过不完全 Cholesky 分解法预处理的系数矩阵。

3.3 BICGSTAB 算法

BICGSTAB 算法的主要思想是：对于 *n* 阶线性方程组 *Ax=b*,令初始近似解为 x_0,第 *k* 次近似解为 x_k，相应的第*k*次残差 $r_k = b - Ax_k$，有 *k* 阶 Krylov 子空间 $K_k=\mathrm{span}\{r_0,Ar_0,\cdots,A^{k-1}r_0\},\tilde{K}_k=\mathrm{span}\{\tilde{r}_0,A^T\tilde{r}_0,\cdots,(A^{k-1})^T\tilde{r}_0\}$。该算法在Krylov子空间$K_k$中选取序列$p_k$，通过选择合适的参数 α_k 和 β_k，产生 x_k 和 r_k，且满足 $r_k\in x_0+K_k, r_k\perp\tilde{K}_k$,同时，计算的残差 r_k 总是取最小范数。有：$r_{k+1}=r_k+\alpha_k p_k$，$p_{k+1}=r_{k+1}+\beta_k p_k$。

在求解实际问题中，BICGSTAB 算法通常与预处理技术相结合以提升收敛速度，增强稳定性。流程如下：

a) 给定系数矩阵*A*，向量*b*，初始值x_0，最大迭代次数k_{max}，最大容许误差 ε_{max} 和预处理矩阵*M*。则初始残差$r_0=b-Ax_0$，令 *k*=1, $\tilde{r}_0=r_0$。

b) 若 $k\le k_{max}$ 且 $\varepsilon\ge\varepsilon_{max}$，则跳转c)，否则，停止并输出 x_k。

c) $\rho_{k-1}=\tilde{r}^T r_{k-1}$。若 $\rho_{k-1}=0$ 或 $\omega_{k-1}=0$，算法终止，输出失败信息，否则转d)。

d)若 $k=1$,则$p_1=r_0$，否则，计算 $\beta_{k-1}=(\rho_{k-1}\alpha_{k-1})/(\rho_{k-2}\omega_{k-1}), p_k=r_{k+1}+\beta_{k-1}(p_{k-1}-\omega_{k-1}V_{k-1})$

e) 求解方程 $M\hat{p}=p$，计算 $V_k=A\hat{p},\alpha_k=\rho_{k-1}/(\tilde{r}^T V_k)$，$s_k=r_{k-1}-\alpha_k V_k$。

f) $\varepsilon=\|s_k\|$，若 $\varepsilon\ge\varepsilon_{max}$，则 $x_k=x_{k-1}+\alpha_k\hat{p}$，否则，停止输出 x_k。

g)求解 $M\hat{s}=s$，$t=A\hat{s}$，$\omega_k=t^T s/(t^T t)$，$x_k=x_{k-1}+\alpha_k p_k+\omega_k\hat{s}$，$r_k=s-\omega_k t$，$\varepsilon=\|r_k\|$，令 $k=k+1$，转 b)。

3.4 不完全 LU 分解

运用不完全 LU 分解获取式(1)的预处理矩阵 *M*。若矩阵 *A* 的顺序主子式都非奇异，则一定能进行 LU 分解。若取预处理矩阵直接进行 LU 分解，得 *M*=*LU*，则理论上最优，但在实际应用中并不可行，一种不完全 LU 分解是指把系数矩阵 *A* 分解为 $\tilde{L}$ 和 $\tilde{U}$，它们的两个因子分别与 *A* 的下、上三角部分有完全相同的非零元结构，这种分解就是无填充的不完全 LU 分解，记为 ILU(0)，如式(7)所示。“0”代表因子分解是产生的非零元素的个数为 0。

$$A\approx M=\tilde{L}\tilde{U} \tag{7}$$

4 CUDA 平台的实现

4.1 基于 HMEC 格式的算法

HMEC 格式的数据结构由多个 ELL 部分和一个 CSR 部分组成,其在 CUDA 平台上的 SpMV 算法由若干个 ELL kernel 和一个 CSR kernel 组成。如图 5 所示，在计算过程中，先将存储为 HMEC 格式的稀疏矩阵 *A* 和向量 *x* 从 host memory 中传输到 global memory,再依次启动这些 kernel，计算完成后，将结果从 global memory 传回 host memory。在整个算法执行过程中，ELL 所需启动的线程数随着剩余矩阵行数的减少而减少。

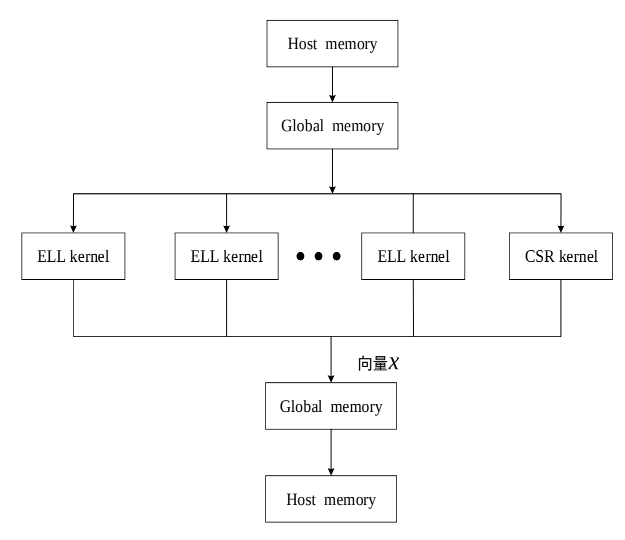


图 5　基于 HMEC 格式的执行算法

对于大型稀疏线性系统，其系数矩阵 *A* 是稀疏的，用 Matlab 对矩阵 *A* 进行处理，以 ELL kernel 为例，可得对应格式的值、列、用于记录矩阵 *A* 按每行非零元的个数降序排序时产生的行索引 jds_row 及每次划分 ELL 时的非零元行数和阈值 *K*。

运用 CUDA 平台编写的 SpMV 算法如下：

1）ELL kernel：

if (row< ELL 格式的行数)

for (int i = 0,1,2...阈值 *K*)

对应行、列中元素相乘，得到结果

根据行索引数组 jds_row 求得经排序后的结果

2） CSR kernel：

if (row2<CSR 格式的非零元行数)

for (int elem =row_ptr[row2]…row_ptr[row2+1])

对应行、列中元素相乘，得到结果

根据行索引数组 jds_row 求得经排序后的结果

4.2　基于 Cublas library、Cusparse library 的算法

Cublas library 就是在 NVIDIA CUDA 中实现 Blas(基本线性代数子程序)。在使用 Cublas library 时，应用程序必须在 GPU 内存空间中分配所需的矩阵向量空间，并将其填充数据，再顺序调用所需的 Cublas 函数，最后将计算结果从 GPU 内存空间上传到主机中。Cusparse library 包含了一系列处理稀疏矩阵的基本的线性代数子程式，是 CUDA 函数库的一部分，从 C，C++中调用。

本文的 SpMV 求解问题正可以归结为稀疏矩阵与一个稠密向量间的操作，运用 Cusparse library 可以较好地实现。对于大型稀疏线性系统的求解，本文采用了适用于不对称矩阵的不完全 LU 分解预处理 BICGSTAB 法和对称正定矩阵的不完全 Cholesky 分解预处理共轭梯度法，在运算过程中，GPU 负责迭代前和每次迭代过程中的矩阵与向量，向量与向量间的并行计算，CPU 负责迭代循环和收敛条件的控制以及标量相除等操作。

运用 Cublas 、Cusparse 库函数进行求解的函数：

调用 cusparseCreateMatDescr 函数创建 descr_L，descr_U，descr_A，并调用 cusparseSetMatIndexBase 与 cusparseSetMatType 函数进行初始化。

调用 cusparseDcsrsv_analysis 函数生成 info_L,info_U

计算初始残差 $r = b - Ax_0$

调用 cusparseDcsrmv 函数计算 $r=Ax0$

调用 cublasDscal 函数计算 $r =-Ax_0$

调用 cublasDaxpy 函数计算 $r = b+r= b - Ax_0$

5　实验分析

本文的计算平台为 CPU：Intel Core^{TM} i5-6500 @3.20 GHz；GPU：NVIDIA GeForce GTX1050；运行内存（RAM）的大小为 8 GB；系统是：Windows7 64 位，实验环境为：Visual Studio 2012, CUDA8.0。如表 1 所示，展现了测试所用的稀疏矩阵，测试所用稀疏矩阵全部来自 UF Sparse Matrix Collection[20]。

表 1　测试所用稀疏矩阵

名称	行数	非零元数	平均每行非零元数
HB/685_bus	685	3,249	4.74
HB/1138_bus	1138	4,054	3.56
HB/bcspwr07	1612	5,824	3.61
HB/bcspwr10	5300	21,842	4.12
Nasa/barth	6691	26,439	3.95
HB/gemat12	4929	33,044	6.7
HB/bcsstk23	3134	45,178	14.42
HB/bcsstk24	3562	159,910	44.89
HB/bcsstk28	4410	219,024	49.67
HB/bcsstk25	15439	252,241	16.34
HB/bcsstk16	4884	290,378	59.45
HB/bcsstk17	10974	428,650	39.06
HB/psmigr_1	3140	543,160	173

名称	行数	非零元数	平均每行非零元数
HB/bcsstk33	8738	591,904	67.73
HB/bcsstk29	13992	619,488	44.27
Hollinger/g7jac180	53370	641,290	12.01
Boe/crystk02	13965	968,583	69.35
Boe/bcsstk36	23052	1,143,140	49.59
ATandT/twotone	120750	1,206,265	9.99
HB/bcsstk30	28924	2,043,492	70.65
Boe/ct20stif	52329	2,600,295	49.69
Rothberg/3dtube	45330	3,213,618	70.89
Chen/pkustk04	55590	4,218,660	75.89
Chen/pkustk12	94653	7,512,317	79.37
Rothberg/gearbox	153746	9,080,404	59.06
Boeing/pwtk	217918	11,524,432	52.88

5.1　SpMV 计算时间

对于大型稀疏方程组，采用调用 GPU kernel 的方式计算一次 SpMV，其中，HEC 存储格式表示划分一次 ELL，将矩阵 A 划分为 ELL 和 CSR 两部分存储。因实验条件的限制，随着划分次数的增多，采用 MATLAB 进行处理的时间也越长，故划分的次数有合适的阈值 p，因本文所采用的矩阵的行数在六百到二十万的量级间，经本文实验可知，p 取 2 或 3 为宜，这里 p 取 2，即将 HMEC 格式划分 2 次 ELL 格式的方式，计算时间如如表 2 所示。

表 2　单次 SpMV 计算时间(单位 ms)

名称	COO	CSR	ELL	HYB	HEC	HMEC(2 次)
HB/1138_bus	0.019	0.024	0.022	0.026	0.027	0.028
HB/bcspwr07	0.027	0.029	0.025	0.029	0.036	0.029
HB/bcspwr10	0.052	0.031	0.039	0.047	0.045	0.041
Nasa/barth	0.058	0.039	0.046	0.063	0.058	0.048
HB/gemat12	0.062	0.037	0.082	0.035	0.033	0.033
HB/bcsstk28	0.192	0.163	0.167	0.129	0.121	0.112
HB/psmigr_1	2.212	2.054	3.58	1.26	1.871	1.567
HB/bcsstk33	0.592	0.471	0.495	0.425	0.395	0.362
HB/bcsstk29	1.13	0.57	0.698	0.412	0.386	0.335
Hollinger/g7jac180	1.25	0.78	2.61	0.642	0.593	0.568
Boe/crystk02	1.48	0.83	4.53	0.715	0.653	0.614
Boe/bcsstk36	1.65	0.99	1.425	0.874	0.71	0.645
ATandT/twotone	3.127	2.854	7.48	2.536	2.215	1.984
HB/bcsstk30	1.92	1.68	2.089	1.398	1.112	1.034
Boe/ct20stif	2.86	2.52	38.59	2.21	1.88	1.69
Rothberg/3dtube	4.35	3.95	31.14	3.66	3.09	2.97
Chen/pkustk04	3.89	3.16	68.51	3.27	2.62	2.36
Chen/pkustk12	10.68	8.59	95.63	8.25	7.14	6.52
Rothberg/gearbox	2.868	0.952	5.04	3.874	3.498	3.287
Boeing/pwtk	5.914	5.152	12.92	4.967	4.275	3.876

由表 2 可知，虽然 ELL 格式非常适合特征为矢量的结构，但是当稀疏矩阵每行的非零元素值的数目变化特别大时，ELL 格式就会丧失这种高效性。当稀疏矩阵 A 的非零元数较少，即小于一定的阈值 k 时，CSR 存储格式所需的时间最少；当稀疏矩阵 A 的非零元数较多，即大于一定的阈值时，本文提出的稀疏矩阵存储格式所需的时间最少，在本实验中，阈值 k 取 30000 左右适宜。原因是 CSR 格式只需存储矩阵 A 中非零元的值、列和行偏移，当非零元数较少时，存储的数据量也较少，有较快

的计算速度；当非零元数较多时，随着存储的数据量的增多，CSR 格式的求解速度受到了一定的制约，而 HMEC 存储格式因拥有 ELL 格式更善于并行计算的优势，且经多次划分 ELL 格式后，只需存储稀疏矩阵 *A* 中非零元的值和列，存储的数据量较少，拥有更快的并行计算速度。

在本次单次 SpMV 运算实验中，对比于单次划分 ELL 格式，随着矩阵 *A* 中非零元数及大小的增加，HMEC 格式相较于 HEC 格式的优势愈加凸显，对于矩阵 HB/psmigr_1，达到了 16.25% 的加速效果；对比于现有的通用存储格式(COO、CSR、ELL、HYB)，对于矩阵 Boe/ct20stif，本文提出的稀疏矩阵存储格式 HMEC 有最好的加速效果，相较于 CSR 格式达到了 32.93%，而对于

矩阵 Rothberg/gearbox，CSR 格式取得了最好的加速效果，这是由于该矩阵的非零元值都为 1，数据量少，更利于 CSR Kernel 的并行计算。

5.2　不完全 LU 分解预处理 BICGSTAB 法

如图 6 所示，HMEC 采用划分 2 次 ELL 格式的方式，CSR(Cusparse)和 HYB(Cusparse)分别代表调用 cusparseDcsrmv()和 cusparseDhybmv()函数计算 SpMV 的方式，运用不完全 LU 分解预处理 BICGSTAB 法计算时间如图 6 所示。

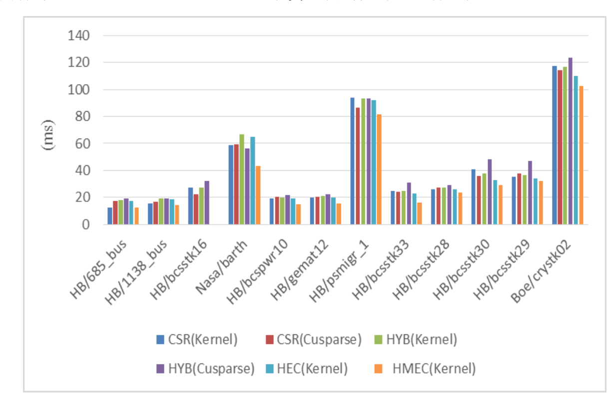


图 6　BICGSTAB 法计算时间

由图 6 可知，本文提出的稀疏矩阵存储格式 HMEC 在进行多次的 SpMV 运算时，有良好的加速效果。特别地，当稀疏矩阵 *A* 为 HB/bcsstk33 时，该存储格式有所有格式中相对最短的 GPU 并行计算时间，为 16.32 ms，相较于 CSR 存储格式加速效果显著，达到了 31.89%。

矩阵 HB/bcsstk33 的非零元数为 591904，远高于矩阵 HB/1138_bus 的 4054，计算时间却相近，这是由于矩阵 HB/bcsstk33 中的非零元值都为整数 1，大大加快了 SpMV 运算的求解速率。

矩阵 HB/bcsstk16 中每行的非零元数都相同，不存在剩余矩阵，因而其只有 ELL 存储格式，无法进行 HYB 等混合格式的划分，表中 NA 代表其数值与 ELL 格式的时间相同，这里，ELL 格式的计算时间与 HYB(Kernel)的计算时间相一致。

5.3　不完全 Cholesky 分解预处理共轭梯度法

目前，不完全 Cholesky 分解预处理共轭梯度法(incomplete Cholesky factorization preconditioned conjugate gradient, ICCG)应用广泛。如在电磁问题中，由有限元单元法离散化导出的线性代数方程组(其系数矩阵记为 *A*)主要有以下基本特征：

a)矩阵 *A* 是稀疏的。网格节点一般是很多的，但每一个节点上的离散化方程只联系几个相邻节点，除与这些节点对应的未知解的系数不为零外，其余系数都是 0，所以，系数矩阵的元素大量是 0，这就是系数矩阵的稀疏性。

b)矩阵 *A* 是对称且正定的。

c) 矩阵 *A* 大都是病态的。若 *A* 的条件数 $cond(A)=(\lambda_{\max}/\lambda_{\min})\gg 1$ ($\lambda_{\max}$ 和 $\lambda_{\min}$ 分别为 *A* 的最大特征值与最小特征值的绝对值)时，称 *A* 为病态矩阵。当 *A* 为病态矩阵时，需用多种方法加速对应方程组的求解。

HMEC 采用划分 2 次 ELL 格式的方式，采用 ICCG 法的计算时间如图 7 所示。

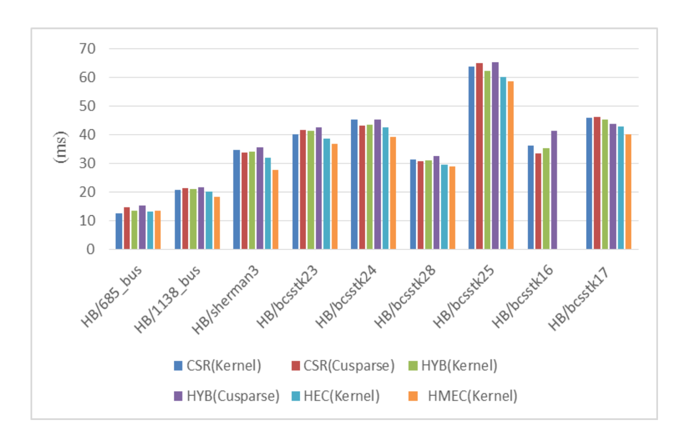


图 7　ICCG 法计算时间

由图 7 可知，本文提出的稀疏矩阵存储格式 HMEC 在运用 ICCG 法求解对称正定线性系统中有着良好的加速效果。当稀疏矩阵 *A* 为 HB/sherman3 时，HMEC 格式在相同矩阵中耗时最短，为 27.85 ms，相较于 CSR 存储格式达到了 17.50% 的加速性能，相较于 HEC 格式达到了 13.29% 的加速效果。

6　结束语

a)本文提出的稀疏矩阵混合存储格式 HMEC 综合了 ELL 和 CSR 存储格式各自的优势，通过不断的划分 ELL 块来降低用于计算的矩阵 *A* 的大小，并利用重排序改善矩阵 *A* 的非零元结构，拥有良好的适应性。

b)HMEC 存储格式在 SpMV 计算中加速效果显著，尤其当矩阵 *A* 较大、非零元数较多时，拥有优秀的加速性能，达到了 32.93%。

c)对于大型稀疏线性系统的求解，当系数矩阵 *A* 为不对称矩阵时，采用不完全 LU 分解预处理 BICGSTAB 法求解，稳定性较高，并达到了 31.89%的加速效果；当系数矩阵 *A* 为对称正定矩阵时，采用 ICCG 法进行求解，算法较于前者更简单，拥有良好的加速效果，达到了 17.50%。

本文从优化稀疏矩阵的存储格式角度来加速大型稀疏线性系统的求解，但进行 SpMV 运算时，采用了单线程计算矩阵 *A* 中一行元素与向量 *x* 中一个元素的乘积，HMEC kernel 函数

仍有进一步优化的空间，如采用以 Half-Warp 为单位进行循环计算的优化方法，以减少线程的空转等待，或使用纹理存储器来存储稀疏矩阵的值，以加速数据读取。此外，本文提及了 ICCG 法在电磁问题分析中的应用，如何使本文提出的加速求解方法成功地应用于电磁问题并扩大其适用范围显得尤为重要，这将是下一步的研究方向。

参考文献：

[1] Al-Mouhamed M A, Khan A H. SpMV and BiCGStab optimization for a class of hepta-diagonal-sparse matrices on GPU [J]. Journal of Supercomputing, 2017, 73 (9): 3761-3795.

[2] Gao J Q, Wu K S, Wang Y S. GPU-accelerated preconditioned GMR ES method for two-dimensional Maxwell's equations [J]. International Journal of Computer Mathematics, 2017, 94 (10): 2122-2144.

[3] Hou G, L Wang. A generalized iterative method and comparision results using projection techniques for solving linear systems [J]. Applied Mathematics and Computation, 2009, 15 (2): 806-817.

[4] Ahamed A C, Magoules F. Iterative methods for sparse linear sys tems on graphics processing unit [C]// Proc of the 14th IEEE International Conference on High Performance Computing and Communication s & the 9th IEEE International Conference on Embedded Softwa re and Systems. 2012: 836-842.

[5] Cools S, Vanroose W. The communication-hiding pipelined BiCG-stab method for the parallel solution of large unsymmetric linear s-ystems [J]. Paral Lel Computing, 2017, 65: 1-20.

[6] Liu Jianxin, Jiang Pengfei, Tong Xiaozhong. Application of BIC-GSTAB algorithm with incomplete LU decomposition preconditi-oning to two dim ensional magnetotelluric forward modeling [J]. Journal of Central South University (Science and Tech nology) , 2009, 40 (2): 484-491.

[7] Anzt H, Gates M, Dongarra J. Preconditioned Krylov solv ers on GPUs [J]. Parallel Computing, 2017, 68: 32-44.

[8] Gravvanis G A, Filelis-Papadopoulos C K, Giannou-Takis K M. Solving finite difference linear systems on GPUs: C-UDA based parallel explicit preconditioned biconjugate conjugate gradient type methods [J]. Journal of Supercomputing. 2012, 61 (3): 590-604.

[9] Mayer J. Parallel algorithms for solving linear systems with spar se triangular matrices [J]. Computing, 2009, 86 (4): 291-312.

[10] Naumov M. Incomplete LU and Cholesky preconditioned it-erative methods using CUSPARSE and CUBLAS [R]. Santa Clara CA: NVIDIA Corporation, 2011.

[11] Nvidia. CUDA CUBLAS library [EB/OL]. [2012-07-01]. http://developer. download.nvidia.com/compute/DevZone/docs/html/CUDALibraries/doc/CUBLAS_Library. pdf.

[12] Nvidia. CUDA CUSPARSE library [EB/OL]. [2012-07-01]. http://developer.download.nvidia.com/compute/DevZone/docs/html/CUDALibraries/doc/CUSPARSE_Library. pdf.

[13] 殷建. 基于 GPU 的矩阵乘法优化研究 [D]. 济南: 山东大学, 2015. (Yin Jian. Research on performance tuning of matrix multiplication based on GPU [D]. Jinan: Shandong University, 2015.)

[14] Williams S, Oliker L, Vuduc R W, *et al.* Optimization of sparse matrix-vector multiplication on emerging multicore platforms: IBM Research Report RC24704 (W0812-047) [R]. 2008.

[15] Yang Wangdong; Li Kenli; Li Keqin. A parallel computing method using blocked format with optimal partitioning for SpMV on GPU [J]. Journal of Computer and System Sciences, 2018, 92 (3): 152-170.

[16] Buatois L, Caumon G, Levy B. Concurrent number cruncher: a GPU implementation of a general sparse linear solver [J]. Int J Parllel Emerg Distrib Syst, 2009, 24 (3): 205-223.

[17] 袁娥, 张云泉, 刘芳芳, 等. SpMV 的自动性能优化实现技术及其应用研究 [J]. 计算机研究与发展, 2009, 46 (7): 1117-1126. (Yuan E, Zhang Yunquan, Liu Fangfang, *et al.* Automatic performance tuning of sparse matrix-vector multiplication: implementation techniques and its application research [J]. Journal of Computer Research and Development, 2009, 46 (7): 1117-1126.)

[18] Belgin M, Back G, Ribbens C J. Pattern based sparse matrix representation for memory efficient SMVM kernels [C]// Proc of the 23rd International Conference on Supercomputing. 2009: 100-109.

[19] Monakov A, Lokhmotov A, Avetisyan A. Automatically tuning sparse matrix vector multiplication for GPU architectures [M]// High Performance Embedded Architectures and Compilers. Berlin: Springer, 2010: 111-125.

[20] University of Florida sparse matrix collection [EB/OL]. [2017-06-25]. http://www. cise. ufl. edu/research/sparse/matrices/list_by_id. html.